DOCUMENT RESUME

ED 246 842 IR 011 170

AUTHOR Wilson, Mary Alice B.
TITLE Helping Teachers Become Computer Literate: A BASIC Approach. [Third Edition] Inservice Series No. 7.
INSTITUTION Hampshire Educational Collaborative, Northampton, MA.
PUB DATE [83]
NOTE 78p.
PUB TYPE Guides - Classroom Use - Materials (For Learner) (051) -- Audiovisual Materials (100)

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS Adults; *Computer Literacy; *Computer Software; Flow Charts; Independent Study; Inservice Teacher Education; *Microcomputers; *Programing; *Programing Languages
IDENTIFIERS *BASIC Programing Language

ABSTRACT

Designed as a teaching script to introduce adults to computer operation and programming, this manual can also be used for self-instruction on Apple II or Apple IIe computers. BASIC programming language is used in the following lessons: (1) "Running the Machine, Print Command"; (2) "Giving the Computer Information"; (3) "When the Computer Gets Information in a Bunch or Creates Its Own Information"; (4) "Loops and Graphics"; (5) "Random Numbers and High Resolution Graphics"; (6) "Subroutines and Things That Go Peep in the Night"; (7) "Two Dimensional Arrays, Fooling Around with String Variables"; and (8) "Files, Hexidecimals, and Flowcharting." The lessons teach BASIC commands, vocabulary, and programming procedures. Assignments and quizzes are included. (LMM)

**********
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
**********

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

- This document has been reproduced as received from the person or organization originating it.
- ☐ Minor changes have been made to improve reproduction quality.

- Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

ED246842

# Helping Teachers Become Computer Literate: A BASIC APPROACH

## Inservice Series No.7

HAMPSHIRE EDUCATIONAL COLLABORATIVE

IR011170

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Mary Alice B. Wilson
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

## WHAT IS HEC?

In 1974, the school districts in Hampshire County joined to form a new public agency, the Hampshire Educational Collaborative (HEC). At the request of our members, we provide programs and services including vocational education, special education and inservice teacher education. HEC currently operates more than 20 classes serving over 250 students. Hundreds more participate in other HEC sponsored activities and receive service at their own schools.

HEC activities are concentrated in three program areas:

Vocational

- vocational skill training shops
- youth employment project

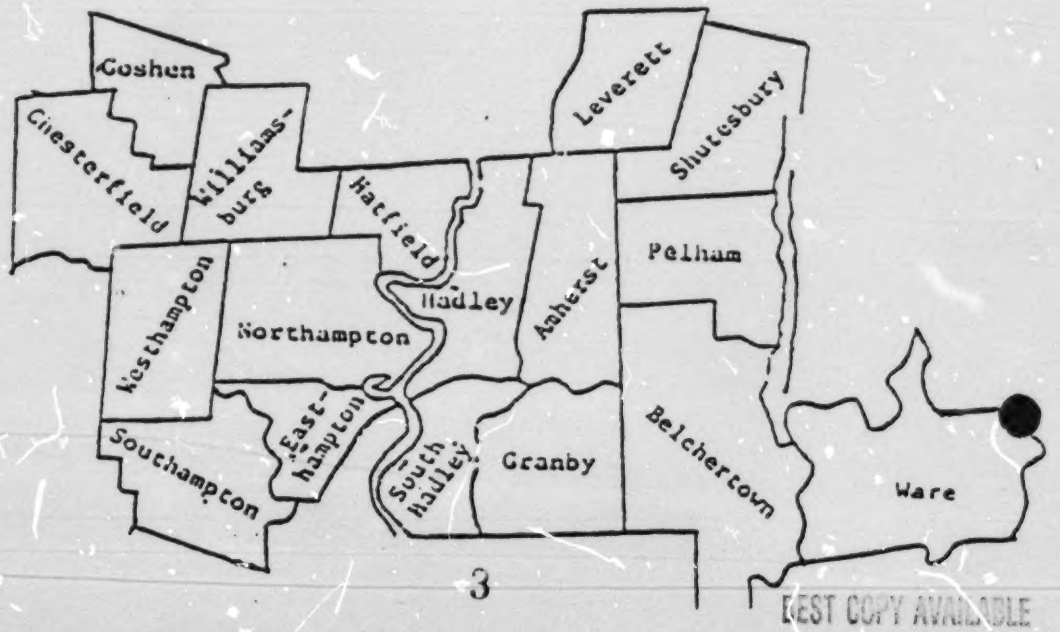
Special Education

- classes for special needs students
- specialist services
- technical assistance for local school districts
- extra-curricular opportunities for special needs students

Staff Development

- resource center activities (seminars, library, recycle materials)
- curriculum development (basic skills, computers, energy, nutrition)
- technical assistance to school system inservice planning teams
- staff development workshops and courses for teachers and administrators
- microcomputer resource center

HEC is governed by a board of school committee representatives from the following towns.






BEST COPY AVAILABLE

TABLE OF CONTENTS


Lesson One: Running the Machine, Print Command 1

Lesson Two: Giving the Computer Information 12

Lesson Three: When the Computer Gets Information in a Bunch or Creates its own Information 21

Lesson Four: Loops and Graphics 29

Lesson Five: Random Numbers and High Resolution Graphics 37

Lesson Six: Subroutines and Things That go Peep in the Night 43

Lesson Seven: Two Dimensional Arrays, Fooling Around with String Variables 51

Lesson Eight: Files, Hexidecimals and Flowcharting 61

ACKNOWLEDGEMENTS

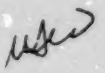
These lessons were developed from our experiences teaching computer programming at the Hampshire Educational Collaborative through grants from E.S.E.A. Title IVC. I would especially like to thank the teachers in Project LOGIC for their dedication and good humor and the teachers participating in Project Capture for their willingess to fill in an incredible number of evaluation forms. Although there are literally so many people who have helped make this work possible that it is inpossible to list them all, I must mention at least the superintendents of the LOGIC sites, Paul Lengieza and Donald Buss, principals Linda Suriano, Dot Ladd and John Pipczynski, HEC Director Earle Batchelder, and State Department of Education staff Al Trocchi and Jack Reynolds. Thank you.

Although I recognize that I must take the responsibility for the inevitable misakes in these lessons, I would like to thank the following people who have helped me develop this approach to teaching programming: Clark Adams, Bruce Brady, Hy Edelstein, Rick Horlick, Trina Hosmer, Len Huber, Alan Lipp, Bill Masalski, George Smith, Bob Sullivan, Howard Wailgum, Bill, Beth and Cathy Wilson.

There have been pages like this is the past, when I have mentioned the importance of my daughters Cathy and Beth for their moral and emotional support. Although that support remains undiminshed, I would also like to thank them for their practical advice in the pedagogy and development of computer programs. It is to them, the younger students just beginning to explore the computer and their teachers that these lessons are dedicated.

In this third revision of the text, I would also like to thank the staff of the Springfield Regional Education Center and the Eashampton, Northampton and Amherst Firemen who have used and helped to revise this edition.

Spring 1983



BASIC

LESSON ONE: RUNNING THE MACHINE, THE PRINT COMMAND

A. Care and feeding of the APPLEII

1. Sequence for starting/stopping

a. Check that the computer (typewriter-looking machine, and the monitor (tv screen) are plugged into sockets and to each other. The disk drive (black box) should be plugged into the computer.

b. Turn on the monitor.

c. Open door on Disk Drive (flips up from the bottom).

d. Insert diskette labelled "DOS 3.3 System Master" into Disk Drive, label side up and at the front. (DOS = Disk Operating System). You can write progams in BASIC on the Apple without loading the System Master Disk (or a file disk). However, you will not be able to save the programs you write.

e. Close the Disk Drive door.

f. Turn on the computer (back left)

g. The light on Disk Drive will come on, there will be whirring noise, wait. Eventually everything will quiet down. If you have waited for 2 minutes, something is the matter. Press the RESET key (on some computers you will need to press the CTRL and RESET keys simultaneously.) After the light goes out check that you have loaded the right disk. Go through the process again. DO NOT turn off the computer or open the disk drive door while the light on the Disk Drive is on. (To understand why, get a flashlight, watch what happens when the disk drive door closes. That piece that comes down is the HEAD which does the electronic reading of the information on the disk).

2. To turn off the machine:

a. Turn off the monitor (and printer if it is on too.)

b. Open the disk drive door (to avoid having an electrical surge damage the disk.)

c. Turn off the computer.

d. If there is danger of electrical storm or surge, unplug the computer.

e. If you are using your own disk which you have not labelled, do so now (use a felt tipped pen not ballpoint or pencil which may damage

the surface of the disk).

f. Put the disk(s) you have been using in a safe place. Unsafe=where there is dust, cigarette smoke, heat, magnetic field, chance of being bent or being marked on: for example inside a parked car (heat), in a cabinet with magnetic catch, near machines that use alot of electricity, in a stack of important stuff.

B. Preparing your own disk

In order to save programs you write, you will want a specially prepared disk:

1. Load the System Master (see A.1. above)

2. Take a blank disk (you can check to make sure that nothing is on the disk by inserting it in the disk drive and typing CATALOG (press the RETURN key). It is possible to reuse an old disk for this purpose, just make sure that whoever used the disk before doesn't mind having it TOTALLY ERASED. If the disk is really blank, the computer will whirr indefinitely. Press RESET (of CTRL/RESET.)

3. With your blank (or recycled) disk in the disk drive (door closed), type

INIT HELLO

The Disk Drive light will come on, there will be lots of scrapping and whirring as the computer sets up the disk so that it can be used on this type of computer. You can take this disk to another Apple will the same sectoring system. You cannot use it with a different brand of computer (or a Apple 3.2 unless you have certain extra disks to convert the information.) It also cannot be used with some programs (software) or languages. HOWEVER, it is ready for you to use to save your BASIC programs.

By the way you might want to see the catalog of your new disk. There is one program on it. (Did you realize that you told the computer to initialize the disk (INIT) with a program (actually a blank program) called HELLO? Hello is the traditional name of the first program on every disk. If you are incensed by that, name it something else, just pick a name you won't want to use later.) Type CATALOG. There it is.

C. The keyboard

1. Most of the keys look familiar to those of you who type. Those of you who don't type easily have the advantage since you don't have to unlearn some typewriter rules, but the disadvantage of not being able to find the right keys. It will be frustrating at first, but

you'll learn.

2. The shift key produces the upper character on any key with two characters. It DOES NOT produce capital letters.

3. The RETURN key would be better labelled ENTER. Nothing you type into the computer is read by the computer until you press the RETURN key.

4. The back arrow ← erases.

5. The forward arrow → passes over letters without erasing

6. The CTRL (control) key is always used with another key. It has lots of uses, the most important of which is

CTRL/C (pressing the control key and the C key simultaneously. That combination stops the processing of a program. Very useful.

7. Ø (with a slash through it) is 0, O is the letter after N. Do not mixed them up. Likewise, l is a number, l is a lowercase L. Typists used to using these numbers and letters interchangeably will have to adjust. (They really do look different on the monitor, even if they don't always look different in print. They are treated very differently by the computer.)

D. Terms you will need:

1. cursor: the little flashing square which tells you where the next letter you type will be placed.

2. microcomputer: refers to the central processing unit (CPU) which does the work. Looks something like a typewriter. (It is definitely worth looking inside a computer. Make sure the power is turned off first however.

3. monitor: TV screen also called CRT (cathode ray tube) and VDT (video display terminal). Although all the data is not in on the safety of these units, it is smart of keep yourself a minimum of 18 inches from the screen (both for your eyes and for microwave emissions.)

4. Disk Drive: shoe box with a garage door, referred to as DOS (disk operating system rhymes with hoss). Without DOS, you could only use the Apple to write simple BASIC programs which could never be stored. There is another storage system, cassettes, which is slower than disks.

5. Diskette: thin record-like object which is always kept inside a paper cover. (Actually made from material as thin as cassette tape, it would truly be a floppy disk if it were to be removed from the protective cover.) The diskette is divided into concentric circles (during the initializing (INIT) process. There are 35 concentric circles or tracks, each divided into 16 sectors in 3.3 DOS. Inside the disk drive, the diskette is read from the bottom (opposite

the label) at the place where there is a oblong hole in the paper. If ANYTHING should damage the diskette (pressure from a pen, dust particles, warping from heat) the reading will be messed up. Information on the diskette is added and erased electronically (magnetically). Therefore magnetic fields (magnets or machines generating alot of electricity) can produce elegant garbage.

6. Hardward refers to anything faintly resembling a machine (the computer, the monitor, disk drive, printer).

7. Peripherals is anything physically attached (connected) to the computer (monitor, printer, plus things like game paddles, graphics tablets, telephone modem connectors).

8. Software is anything on a diskette, i.e. a program or group of programs. All the games, instructional programs, data based management systems are software. Certain kinds of software may also include peripherals (for example a program which uses both a disk and a graphics tablet.) Like everything else in this world there is The Good (excellent, incredible) Software and The Bad (insulting, poorly designed, unworkable Software. Virtually everyone using a computer uses software designed by other people (in addition to personally developed programs). It is, therefore, very important to be a good comsumer, judging your needs and matching them with the best product.

9. Memory. The ability to store information. There is both the active memory of the computer (how much information space is available while you are working at the machine or your WORKSPACE) and stored memory. The stored memory is simply the ability to put information on disks (or cassettes). The critical selling point of a computer is how much active memory is available. (A 48K computer has more memory that and 32K etc.) There is other jargon related to memory that will be dealt with later.

E. Programming

At last. A program is a series of statements or instructions which the computer carriers out (executes) in sequence.

1. If there is anything on your screen, you may want to clear it by typing HOME (anything you type must be followed by pressing the RETURN key, so assuming you will remember to do that, especially after you sit there for 5 minutes waiting for something to happen, it won't be mentioned again.)

2. Type in the following:

```
NEW
```

```
10 REM PROGRAM TO DEMONSTRATE PRINT COMMAND
20 PRINT "GOOD MORNING"
30 END
```

Press the RETURN key at the en of each line. However, in line 10, do not press the RETURN key until you have typed the "d" in COMMAND. Don't worry when you get to the right edge of the screen, just have faith and keep typing. Check now that you have typed zeros (0) instead of Os (o) in the line numbers. The zero has a slash through it. Likewise, make sure that you have used the one, not the L in typing the 10. If you have made any mistakes, retype that line (The computer will resequence them for you. When you are finished the new line and have pressed RETURN, type LIST (and press RETURN) to see the program re-sequenced.

Now, what is going on:

```
NEW
```

This clears the memory of any previous programs. If you do not do this, you will get your new program mixed up with whatever happens to be in the computer. (To find out what the computer has as its program, type LIST).

```
10 REM PROGRAM TO DEMONSTRATE PRINT COMMAND
```

REM stands for "remark". It is not a command for the computer to do anything. It is designed to help the computer programmer (you) figure out what you are doing in the program and why. Remark statements provide the "documentation" (description for the program reader) of the program.

```
20 PRINT "GOOD MORNING"
```

The 10 and the 20 are line numbers. Every line in a program MUST have line numbers (so the computer knows what to do next.) Traditionally programmers skip 10 line numbers so that when they want to add lines later they can easily insert them.

PRINT is a command to the computer to write whatever is inside the quotation marks on the monitor screen.

```
30 END
```

Guess what that means?

3. Type

```
RUN
```

(Notice that there is no line number in front of RUN. RUN is not part of the program, but a command directly to the computer.) On the line below RUN should be the words GOOD MORNING.

4. OK, you are in business. You can change the print line by typing in a line with the same line number (20), you can delete a line by typing the line number (for example 30) and then pressing return (i.e. an empty line). You can add lines, anywhere in the sequence of the program, for example, 21, 22, 23, 24 etc. The computer will put them in the right place. (If you do that, you will want to type LIST occasionally so that you are able to read what you have written in the right sequence.). Add at least 3 lines of PRINT to the program. Use the syntax:

## PRINT "xxxxx" where ## is the line number and xxxx is what you want to have printed.

F. Carriage control

1. Type in the following program

```
NEW
10 REM PROGRAM TO DEMONSTRATE USE OF COMMAS AND SEMICOLONS
20 PRINT "HELLO",
30 GOTO 20
40 END
```

Now type RUN. Don't panic, you can always stop the program by pressing the CTRL and the C key simultaneously. What does the GOTO statement do? What does the comma in line 20 do?

2. Retype line 20 as:

```
20 PRINT "HELLO";
```

Run it. What does the semicolon do? To get a space between the hellos, retype as:

```
20 PRINT "HELLO ";
```

3. In addition to the comma (which places things at columns 1, 17 and 33 on the monitor) and the semicolon which puts them right next to

each other (by supressing the carriage return completely), there is also the tab.

```
NEW

 10 REM PROGRAM TO DEMONSTRATE TABBING

 20 PRINT TAB (5) "LEFT"TAB (30) "RIGHT"

 30 GOTO 20

 40 END
```

When calculating tabs, there are 40 spaces on the screen. (Actually, you can TAB to 255, but that is not terribly useful unless you are printing something on very wide paper.) Note that the 5 and the 30 with the TAB commands measure from the left hand margin, not from the last word printed.

For those of you familiar with the PRINTUSING command on some other computers, Apple doesn't have it. Sorry. It also has a very poor line editing system. There is an system for moving the cursor around, but it is so awful, most people find it easier to just retype a line.

4. One last change type:

```
25 PRINT
```

and run the program. The empty print statement can be very helpful in keeping what you have printing from looking too scrunched up on the screen. (You can get two lines either by added a 26 PRINT, or by doubling the line (25 PRINT : PRINT)

G. Screen images

1. Add a line to the previous program

```
15 INVERSE
```

Run it.

2. Now try

```
15 FLASH
```

Run it.

3. To get rid of all that stuff either type

```
15 NORMAL
```

or just remove line 5 all together by typing

15 (press return)

These screen changes allow you to make one line inverse or flashing and then return to normal printing. Remember if you want to clear the screen between programs (or during a program) the command is

HOME

Try the previous program with

15 HOME

(The clearing of the screen happens so quickly that you may not see it. There are situations where it is very useful, however.

H. Saving programs

Now that you have your own diskette ready for use, and are about to write some exciting programs, it is time to save some of your work. Notice that each time you typed NEW, you erased the previous program you were working on. You also do not name programs when you begin them, you name them only when it is time to save them.

1. Select a name for the program you have been working on. Chose something that will help you remember what it is about when you want to find it again. The name must begin with a letter (not a number), not include any commas, and be no more than 30 characters long (most are much less). You may not use any of the words the computer reserves for itself (like PRINT, HOME, LIST.) The name of the file can have numbers in any place but the first place. If you select a name you have already used, when you type Save, THE OLD PROGRAM WILL BE ERASED.

2. Put your diskette into the disk drive (label front and up, close the door).

3. Type

SAVE TABDEMO (or whatever you are calling it)

lights, noise and it is done. (Remember do not open the disk drive door, or do anything while that disk drive light is on.) To make sure it is there you might type CATALOG, or reload the system (open the door, turn off the machine, make sure you disk in still in the disk drive, close the door, type RUN TABDEMO (or whatever you named it.) It really works doesn't it?

4. It is important to remember to re-save a program after you have made changes in it. Most of us learn by typing in a program which we like, saving it and them making wonderful, inspired changes,

forgetting to save the new version, turning off the machine and, the next day, showing our wonderful new program to a friend. There isn't any wonderful new program, there is only the nasty old program. Moral. After you make changes SAVE the program (either under the same name, if you only want the new version, or under a different name if you want both versions.)

I. Commands you have learned:

NEW

Clears the active memory of previous programs

REM

Remark: ignored by the computer, useful to help the programmer know purpose of a section of a program.

PRINT " "

Command to have computer write whatever is between the quotation marks on the screen. PRINT alone produces a blank line.

END

Command indicating the program is over.

RUN

Used alone, the computer will execute the program which has just been written. Used with a program name (As in RUN TABDEMO) the computer will load the program named from the disk and execute it.

INIT

Command to initialize (prepare for use) a new diskette

GOTO

An unconditional branching command, the computer goes to the line indicated rather than to the next line.

,

Used in PRINT commands to tab to the 1, 17 and 34th columns on the screen.

;

Used in PRINT commands to keep the cursor from going to the next line (to supress the cursor movement).

TAB (#)

Print command to move the cursor the number of spaces indicated in the parentheses counting from the left side of the screen

INVERSE

Print command to produce dark on light letters on the monitor

FLASH

Print command to produce flashing (inverse, normal, inverse, etc) letters on the monitor

NORMAL

Print command to return from inverse or flash

HOME

Command to clear the screen, can be used directly or in a program

SAVE xxxxx

Command to store a program named xxxxx onto a diskette

J. Assignment

1. Write, run and save a program which fills the entire monitor screen with words. Try using as many of the commands you have learned as possible.

2. Write, run and save a program which prints some kind of pattern. Try to use as many of the commands that you have learned as possible.

3. Review both the commands and the terminology presented until you feel comfortable with them (can touch, describe, or use them without hestitation.)

REVIEW QUIZ, BASIC, CHAPTER 1

sequence for loading (booti[illegible]g) a program

sequence for turning off th[illegible]e computer

care of the diskette

| keys: | com[illegible]mands: |
|---|---|
| shift | RU[illegible]N Programname |
| return | RU[illegible]N |
| back arrow | SA[illegible]VE Programname |
| forward arrow | LI[illegible]ST |
| ctrl/c | CA[illegible]NC |
| cursor | LIST |
| microcomputer | HOME |
| monitor | NEW |
| disk drive | REM |
| diskette | PRINT " " |
| CPU | PRINT |
| CRT | END |
| DOS | GOTO |
| VDT | , |
| hardware | ; |
| perpherials | TAB ( ) |
| software | INVERSE |
| memory | FLASH |
| program | NORMAL |

# BASIC

## LESSON TWO: GIVING THE COMPUTER INFORMATION

A. Giving the computer information from the keyboard

1. Type in the following program

```
NEW

10 REM PROGRAM TO DEMONSTRATE INPUT STATEMENT STRING AND
NUMERIC VARIABLES

20 INPUT "GOOD MORNING WHAT IS YOUR NAME? ";NA$

30 PRINT:PRINT

40 PRINT "I LIKE THE NAME ";NA$

50 END
```

OK, what is happening?

```
NEW
```

You remember that; it clears away any program left in the memory so you can start anew.

10 REM, a statement ignored by the computer, but put there to help the programmer know what is going on. This one is so full of jargon it is hardly a help. Never fear, it will be clear in a minute.

```
20  INPUT "WHAT IS YOUR NAME? ",NA$
```

The input statement, like the print statement will print whatever is between the quotation marks. HOWEVER, the input statement tells the computer that there is going to be data (information) entered in the computer and that, before the line is finished, the computer is going to know the name of the variable to which the information will be assigned. In this case the variable (or box or memory location) will be called NA$. It really doesn't matter what the first letters are, it does matter that the last character is the $ (called string) because that tells the computer that the value should be stored as a series of alphabetic characters not as a number. So the computer prints what is inside the quotation marks then waits for the input (NA$). When you type in Jane Doe, it gives the variable NA$ the value of Jane Doe.

```
30 PRINT:PRINT
```

Two empty carriage returns, only there to make the printing on the

screen look classier.

```
40 PRINT "I LIKE THE NAME ";NA$
```

The PRINT statement you remember. The space after the e in name keeps the space between words. The ; supresses the carriage return so that the cursor will not move forward, but immediately print the next thing, which in this case is the value of NA$ (Jane Doe or whatever).

```
50 END
```

program complete. Run this program a number of times, typing in different names each time. You need to convince yourself that the NA$ box will hold any value you assign to it.

2. Try writing another program which takes string (word) input and prints it out. For example, you could revise one of your pattern programs from last lesson to print someone's name all over the screen. Change the name of the variable to something else (the computer only really reads the first two letters, so don't use both NAMES and NAPS. Also don't start the name of a variable with a number and don't use a command (like PRINT) for a variable name. Remember to put on the $.)

3. The other kind of variable is a numeric variable (number). Try making the following addition to the program above:

```
50 INPUT "WHAT IS YOUR FAVORITE NUMBER?";NU

60 PRINT: PRINT

70 PRINT "WELL ";NA$;", I LIKE ";NU; ", BUT MY FAVORITE NUMBER
IS "; NU*NU;"."

80 END
```

In line 50 the computer is told to wait for a value for the variable "NU" AND that it will be a numeric variable (i.e. there is no $ on the end).

In line 70, the computer prints both the string variable and the numeric variable. It even multiplies (*) the number by itself and prints that number.

4. That's it. That is what the computer does each time information is typed into an interactive program (where the user is asked to respond during the program). Either it takes the information as a word or as a number. It stores it under the name of a variable and does whatever you, the programmer, asked it to do with that information.

Try writing some short programs which ask for both words and numbers.

B. Sequence of operations.

1. Since someone is immediately going to want to do fancy mathematical operations with any number entered we need to do two things: assignment statements and sequence of operations.

```
NEW
10 REM PROGRAM TO DEMONSTRATE ASSIGNMENT
20 INPUT "TYPE IN A NUMBER";X
30 PRINT:PRINT
40 LET PROD=X*2
50 LET QUOT=X/2
60 PRINT "DID YOU KNOW THAT YOUR NUMBER MULTIPLIED BY 2 IS
";PROD;"?"
70 PRINT:PRINT
80 PRINT "AND THAT YOUR NUMBER DIVIDED BY 2 IS ";QUOT;"?"
90 END
```

Not much you don't understand here. In BASIC the unknown is always on the left-hand side. This will take those who frequently use formulas a few moments to get used to. For those whose minds are uncluttered by x and y, it is very easy.

2. Now the issue of precedence (what gets done in what order). The sequence in BASIC is

1st ↑ exponentiations, left to right

2nd */ multiplications and divisions, left to right

3rd +- additions and subtractions, left to right

As in non-computer calculations, you can avoid precedence by surrounding statements you want completed first with parentheses.

Examples to think about are X=12/6+2, compared to X= 12/(6+2) and so on.

By the way, you can use the computer as a calculator either by writing

a little program

```
NEW

10 ANS=10+3/4

20 PRINT ANS

30 END
```

(A program is called "deferred execution",i.e. the computer doesn't do anything until the completed program is run. You can also use the computer as a calculator for "immediate excution" without program lines, the computer will do instant calculations for you:

```
PRINT 10+3/4
```

try it. It is a rather expensive (and rather bulky) pocket calculator, but very useful.

3. Historical artifacts, or whatever you call the process of using old words for new purposes. As you already know, the RETURN key is realy misnamed. It should be ENTER, but computer designers wanted to keep as many familiar keys as possible one the keyboard. The word variable is another such word. Most books on programming refer to a variable as a pigeonhole, mailbox, empty container, or memory location. All of those words are really easier to use than variable. When you start to write a program, the computer is waiting with lots of empty memory locations. When you name one of those locations (ANS, SUM, whatever) it has a memory location with a name and a value of 0. When you assign the location a value (3), the 3 goes in that spot until you change the value (to 5). If you have called the memory location NA$, the computer expects letters which it will store in the location. If you assign the value "Rumplestilskin", that will be the value of NA$ until you change the value (to "Jane Doe"). Thinking in terms of containers or pigeonholes or mailboxes, is not just a cute illustration, it is really an accurate analogy.

C. Using the Printer

1. Time to start making "hardcopy" of your programs. Given the computer use of HARD and SOFT, you might expect "hardcopy" to be something written in stone (or at least plastic). However, "hardcopy" means paper (remember, that's the stuff we used to write on using something called a pen?). Take your disk with your saved program to a computer that has a printer connected to it. Make sure the printer is on, that the paper is feeding properly and that (if there is such an option) that the printer is "on line". If you have questions about getting the printer ready, check with someone or take the time to read the manual supplied with the printer. Load the DOS system Master (if you don't remember how, re-read BASIC Lesson One, when the disk drive light is out, put in you diskette and load the lesson you want printed by typing

LOAD xxxxxxx (whatever the name of your file is)

2. Now type

PR#1 (printer may make a noise, monitor may stop functioning)

LIST (it should list out the program onto the paper)

RUN (the program will run on the printer. If you program has INPUT statements, you will have to read the instructions from the printer -you probably have them memorized anyway- and type the required input onto the keyboard)

There will be some device on the printer to feed the paper forward so you can tear off the pages you have printed. Be sure to turn if off when you turn off the computer and monitor.

D. Review of Terms

string variable

Indicated by $ at end of the variable name, stored by the computer as alphabetic (also called alphanumeric) characters.

numeric variable

No special indicator on variable name, stored by the computer as a number (and therefore can be used in arithmetic operations.)

variable name

First character must be alphabetic, only first two characters are read by computer.

TYPE MISMATCH ERROR

Some mixup between numeric and string variables. For example assigning a numeric variable a letter value (input "what is your favorite number? "x and the user types A; or asking the computer of add x + NAME$.

bug, debug

No self respecting programmer ever writes a program that runs perfectly the first time. The process of locating the problems, whether by the typing, syntax or logic is called debugging.

syntax

The computer is programmed to expect certain words (like NEW, PRINT etc.). If the spelling of any of the words is not what it expects, the "syntax" is wrong (hence "syntax error". It also expects things to happen in a certain order: after LET it expects the name of a variable and the an =. If it finds a +, things are out of sequence (syntax error).

slots

Without slots the APPLE might still be made in a garage in Silicon Valley. If you look inside the APPLE (power off) there are a series of slots at right angles to the back of the computer labelled 0 to 9. Each of these slots will take an electronic board or "card" (sometimes called "firmware" since it is neither "hard" or "soft"). Usually slot #6 has the board (or interface card) that connects the computer and the disk drive. Slot #1 usually has the interface card for the printer. The command PR#1 sends all output to whatever is connected to #1(printer), PR#0 sends it back to the monitor. PR#6 sends it to the disk drive, which effectively starts the computer reading whatever disk is in the disk drive)

Sequence of operations

If you are going to do any even slightly elaborate calculations with the computer, you will need to memorize (or have easily available to you) the order of precedence used by the computer (↑,*/,+− at each level proceding from left to right)

Interactive

In the old days of computer cards, the computer programmer gave the computer operator a stack of sequenced cards (each card had a program line or a line of data). The cards were read into the computer, whatever was supposed to happen did (or didn't), and was printed out. The programmer returned got his print-out and jumped up and down, either with joy or frustration. If you are part way through a program and type a line without remembering the line number the computer immediately tells you the problem. One level of interaction is between the programmer and the computer. A second type of interaction is between the user and the computer (when someone sitting at the computer is asked "What is your name?".

E. Commands you have learned

INPUT B$

Command, tells the computer to expect a value for the variable which is listed in the statement (in this case, B$.)

INPUT "  " ;B

Command, tells computer both to print what is between quotation marks and to expect variable (in this case, B.)

?

Prompt symbol used by computer when it is expecting input and the INPUT statement does not have " "

$

Last character of string variable

LET A =

Assignment statement, value on right side of = is assigned to box indicated by name (A) on left side of =

↑

Symbol of exponentiation

*

Symbol for multiplication

/

Symbol for division

+

Symbol for addition

-

Symbol for subtraction

( )

Symbol to change order of precedence of arithmetic operations (calculations inside the parenthesis will be done first)

PR#1

Send output to peripheral connected to slot #1 (usually the printer)

PR#0

Send output to the monitor

F. Assignment

1. Create a mnemonic which will help you remember the order of precedence of operations.

2. Write, run and save a program which uses input of string variables and produces a classy looking output.

3. Write, run and save a program which uses input of numeric variables and produces a classy looking output. (You can combine #2 and #3 if you wish.)

4. Print out one of the programs you have written (both the program (LIST) and the output (RUN).

Quiz on Lesson 2

string variable bug,debug

numeric variable slot #1 (etc.)

INPUT interactive

$

?

:

sequence of operations

↑

*

/

+

-

( )

Let

=

hardcopy

PR#1

PR#0

sequence for getting hardcopy

rules for variable name

TYPE MISMATCH ERROR

SYNTAX ERROR

LESSON THREE: WHEN THE COMPUTER GETS INFORMATION IN A BUNCH, OR CREATES ITS OWN INFORMATION.

A. Reading Data

There are many situations where you would not want to type in the information (data) every time the program was run. It is possible to type the data (both string and numeric) directly into the program.

1. Type

```
NEW

10 REM PROGRAM TO DEMONSTRATE READ/DATA STATEMENTS

20 REM F$=FIRST NAME, L$=LAST NAME, N$=STUDENT NUMBER

30 PRINT "CLASS LIST FOR COMPUTER PROGRAMMING I"

40 PRINT:PRINT

50 PRINT "FIRST NAME" TAB (15) "LAST NAME" TAB (35) "ST.#"

60 PRINT

70 READ F$,L$,N$

80 IF F$="XXX" THEN GOTO 1000

90 PRINT F$ TAB(15) L$ TAB(35) N$

100 GO TO 70

500 DATA MARIE,CURIE,88,NAPOLEON, BONEPARTE, 1812,
NICELY,NICELY,711

600 DATA XXX,X,X

1000 END
```

Run the program, now let's go through it to see what happens:

You should have no trouble with the original print headings

```
70 READ F$,L$,N$
```

READ tells the computer that somewhere in the program will be some lines marked DATA. These lines can be ANYWHERE; they are traditionally put at the beginning or at the end where they are easy to find. On Line 60 the computer is looking for 3 string variables. (Actually the student number could have been treated as a numeric variable. However, if there is any chance of the identifying number beginning with a zero (as social security numbers do) or having

letters or blank spaces (like license plates), it is better to treat identifying numbers as strings. Besides, when was the last time you added, subtracted or divided by a student number?)

```
80 If F$="XXX" THEN GOTO 1000
```

IF is a wonderfully powerful command which allows you to test the truth of an expression. Does one side equal the other(=); is one larger (>) or smaller( < ) or are they not equal (<>)?

In this case, the last line of data is put in as an "end of file flag". Since the computer is reading the data in bunches of 3, there MUST be three pieces of data even for the last reading, or the computer gets hysterical: OUT OF DATA. If no end-of-data flag is put in it will get equally unhappy. Notice that the value of the string variable (xxx) is put in quotation marks.

This statement is called a "conditional branch" i.e. if the statement is true, the computer will execute the command at the end of the line. If the condition is false, the computer will not execute the command, but will go on to the next line ("drop through to the next line").

I used 1000 to make sure I didn't get in my own way.

PRINT and GOTO

These statements should be no problem. Notice that after the first READ statement, the computer left a marker after 88, and in the next time through the READ statement, it began with the next piece of data (Napoleon). If you wanted the computer to just read the first three pieces of data over and over, you could add 85 RESTORE (Try it). RESTORE moves the marker back to the beginning of the data.

```
500 DATA MARIE, CURIE, 88, etc.
```

NOTE that there ARE the commas between each piece of data and NO comma at then end of the line. All the data could have been put in one data line, however, moving the "end of data flag" into a line of its own, allows me to add extra students much more easily. Also note that the data "comes out right". The computer has been asked to read data in groups of three, and the data is very properly lined up waiting to be read in multiples of three.

B. ITERATION: when the computer generates its own data

I am indebted to Bill Masalski for the idea for this program.

Type in

```
NEW

10 REM PROGRAM TO DEMONSTRATE ITERATION

20 LET P$="POTATOE"


30 N=N+1

40 IF N=4 THEN GOTO 500

50 PRINT N;"  ";P$

60 GOTO 30

500 PRINT N

1000 END  "
```

After you run this program it is important to "walk "through it. I find columns help figure out what is happening in a program. Try it.

| line number | N | P$ | if true? | printout |
|---|---|---|---|---|
| 0 | 0 | (blank) | | |
| 20 | | potatoe | | |
| 30 | 1 | | | |
| 40 | | | no | |
| 50 | | | | 1 potatoe |
| 60 | | | | |
| 30 | 2 | | | |
| 40 | | | no | |
| 50 | | | | 2 potatoe |
| 60 | | | | |
| 30 | 3 | | | |
| 40 | | | no | |
| 50 | | | | 3 potatoe |
| 60 | | | | |
| 30 | 4 | | | |
| 40 | | | yes | |
| 500 | | | | 4 |
| 1000 | end | | | |

Another way to see what the program is doing is to type TRACE (direct command, no line number), then run your Potatoe program again. All those funny numbers are the line numbers listed as the program is executed. Sit there and follow it until you understand what is happening. If your back and eyes ache, print out a run with the TRACE command on, (also a list so you know what each line number says). To get rid of TRACE type NOTRACE. Tracing your program is a wonderful debugging, and just plain learning, tool.

OK. Now add no more than three lines to this program to make it print out:

```
1 potatoe
2 potatoe
3 potatoe
4
```

```
5 potatoe
6 potatoe
7 potatoe
8 more
```

Don't read on until you have tried it by yourself.

Work first to get the 5 potatoe, don't worry about having an entire warehouse of potatoes, you can always stop the accumulation with a CTRL/C. You do not need a new N=N+1, you just need to get back to the one you have.

Now figure out where to put a new IF statement so that it doesn't get lost in some dead space. There is, for example, no way for the computer to read any line placed between 60 and 500. Make sure you understand why. The best way to work on this problem is to enlarge the chart made earlier.

Finally, a little trick. Remember the PRINT:PRINT. The colon allows two commands to be listed with the same line number. In an IF statement, the command beyond the colon will be executed ONLY if the condition is true. So in the original program, line 40 could have been written:

```
40 IF N=4 THEN PRINT N:END
```

You can now rewrite it as

```
40 IF N=4 THEN PRINT N:GOTO 30

45 IF N=8 THEN PRINT "MORE" :END
```

and remove lines 500 and 1000. Make sure you understand what is happening in each line of this program. (Try to predict what the TRACE listing will be, Type TRACE, run your program and see if you are right.)

2. Iteration, or the counting process, is a very powerful aspect of computer programming. I encourage you to write at least two simple programs which included a counting mechanism (A=A+1) and to chart out the action until you are very comfortable with it.

C. Terms

end of file flag

A "flag" is used by the programmer to provide bogus names or numbers to end the processing of the data. An end of file flag is used with an IF statement to signal the end of the data.

OUT OF DATA

Error statement from the computer indicating it has been asked to read more data that it found in the data lines.

Conditional Branching

Statement in which the computer is asked to test the truth of some expression and, if the statement is TRUE, to execute some command. If the statement is not true, the computer will ignore the command and continue (drop through) to the next line. (An unconditional branch simply sends the computer somewhere else in the program i.e. GOTO 20)

Iteration

Or using a counting variable (B=B+1). The variable is incremented by 1 (or whatever amount you choose) on each loop of execution.

D. Commands

READ

Command telling the computer that to get data from the line(s) labelled DATA

DATA

List of values (alphanumeric or numeric), each item is separated by a comma (but with no comma at the end of the line). Usually put at the beginning or end of the program

IF...THEN

Conditional statement, if true the computer will carry out the command, if not true the computer will go to the next line

B=B+1

Counting expression (iteration) used to keep track of the number of

times a loop has been executed.

RESTORE

Normally the computer puts a marker after the last data it reads. The next time it reads data, it begins after the marker. RESTORE returns the marker to the beginning of the data.

TRACE, NOTRACE

Debugging command which displays the line number of each statement executed. NOTRACE returns the computer to regular execution. In certain complex programs, TRACE may cause problems, if it does, just type NOTRACE to get rid of it.

E. Assignment

1. There a lots of nursery rhymes and songs which use patterns like "one potatoe", write a program to print out a traditional rhyme, or make up one of your own.
2. Write a program which reads both string and numeric variables and produces some classy output.

3. Using TAB(X) and X=X+1, write a short program that prints some symbol or word approximately diagonally across the screen. (You can add this to the rotatoe program.)

Quiz on Lesson Three

READ

DATA

IF...THEN

=

>

>=

<

<=

<>

c=c+1

RESTORE

TRACE, NOTRACE

end-of-file flag

OUT OF DATA

conditional branching

iteration

# LESSON FOUR: LOOPS AND GRAPHICS

a. Loop

1. You have already executed loops (or a set of instructions preformed repeatedly until some condition is met). In Lesson One, the GOTO statement, produced an infinite loop that had to be stopped with ctrl/c. In Lesson Three, the IF N=4 described the condition which would end the loop. There is a very neat convention for creating loops:

```
NEW

10 REM PROGRAM TO DEMONSTRATE FOR..NEXT STATEMENTS

20 PRINT "WATCH ME COUNT TO 10."

30 PRINT:PRINT

40 FOR L=1 TO 10

50 PRINT L

60 NEXT L

70 END
```

2. Make up a chart, and follow this loop program through the 10 steps (or at least 5 of them.) Try some different programs using the FOR/NEXT statements. To check your chart, type TRACE and run the program. The computer prints the line numbers as they are executed. To get rid of them, type NOTRACE.

3. There is a nice way to fool around with the counting of the loop. Try these changes in the program listed above.

```
40  FOR L=1 TO 10 STEP 2
```

What do you think will happen? Try it.

```
40 FOR L=10 TO 1 STEP -1
```

Try that. Make other changes to be sure you understand how to use the step option (which is not needed if you only want the increment to be +1).

B. Graphics

1. To put the Apple into the graphics (drawing) mode, you need to type GR (for graphics). You can do this as a direct command (try it) or as a line in a program. To get back to text (writing) mode, type TEXT.

2. The colors available in the GR mode are

0 black
1 magenta
2 dark blue
3 purple
4 dark green
5 grey
6 medium blue
7 light blue
8 brown
9 orange
10 grey
11 pink
12 green
13 yellow
14 aqua
15 white

If you are using a black and white monitor, remember not to set the line color to black or even a dark color or you will not be able to see what you are drawing. By the way, the computer automatically sets the value of any variable to zero (it is up to you in the program to specify any other value). That is a very helpful approach, except in this situation where 0 is black. If you neglect to specify a color, I can guarentee you that you will spend hours debugging a program that won't work only to find that it is working beautifully, the only problem is that it is drawing a black line on a black background.

3. The GR (low resolution graphics screen) can be looked upon as a grid. There are 40 points (boxes) on both the x (horizontal) and y (vertical) axis. 0,0 is at the top left-hand corner, 39,39 is at the bottom right-hand corner.

For those who teach graphing, you may want to think of the graph as Quad I with the y axis reversed. If you feel very strongly that students should always have 0,0 in the customary place (bottom left for Quad I), there are software programs that will convert the screen for you. There are, of course, also arguments for having students deal with different points of origin. It is usually more difficult for adults to make the adjustment than kids.

4. Type in the following

```
NEW
```

```
10 REM PROGRAM TO DEMONSTRATE PLOTTING WITH LOW RESOLUTION
GRAPHICS

20 GR

30 COLOR =13

40 PLOT 10,30

50 END
```

Where do you think the little square which is being put on the screen is going to be? Try it.

5. You can make an infinite number of changes in this program. Make a loop which changes the place of the plot.

It is almost impossible to make program changes while in the graphics mode since there are only four lines of text available at the bottom on the monitor. Type TEXT to get the full screen for writing. If all those crazy characters bother you, type HOME to clear the screen. You can put the three commands you need on one line by typing TEXT: HOME: LIST

5. You can, of course, draw lines by plotting boxes next to each other. However, there is a shortcut which is fun to use.

```
NEW

10 REM PROGRAM TO DEMONSTRATE HORIZONTAL AND VERTICAL LINES

20 GR

30 COLOR = 7

40 HLIN 0,39 AT 20

50 VLIN 0,39 AT 20

60 END
```

40 (as you might guess) draws a horizontal line from column 0 to column 39 on row 20.

50 draws a vertical line from row 0 to row 39 at column 20.

Try it.

6. And try some changes. Make a loop to change the colors. (It will be difficult to appreciate the subtle differences in color on the black and white screen, but you can survive that.)

If the colors change too fast to enjoy, put in an empty loop (i.e. the

computer spends a second processing it which gives you a visual break between drawings.

```
FOR T=1 TO 1000
NEXT T
```

7. Make a double loop so that you can change the lines as well as the colors. Loops must be nested INSIDE each other, they cannot cross over:

```
30 FOR C=1 TO 15
40 COLOR =C
50 FOR P = 1 TO 39 STEP 2
60 HLIN P, 39 AT P
70 VLIN P, 39 AT P
80 NEXT P
90 NEXT C
100 END
```

To slow it down, you may want to add

```
73 FOR T=1 to 500
75 NEXT T
```

After you have tried this, spend time understanding what is happening. Again a chart is probably a good way to do it (decrease the numbers in each loop to make it easier to chart.) Some microcomputers actually set each loop in at a different margin, Apple does not, but you might want to write it out for yourself that way to get a visual sense of the loops.

C. TERMS

loop

Set of instructions performed repeatedly. An infinite loop is performed until the procedure is manually stopped. (In the old days when programmers brought their IBM cards to the little window of the computer room and were being charged lots of money per second for computer time, an infinite loop could become a very expensive nightmare. Now it is usually done for the fun of it.

Loops can be written using branching commands (GOTO), iteration (N=N+1, IF N..) or FOR/NEXT statements.

graphics mode

Most of the screen is devoted to drawing. The bottom four lines are left for text.

text mode

Full screen devoted to text (writing)

x axis

The scale found across the bottom of most graphs (horizontal).

y axis

The scale found along the side of most graphs (vertical).

low resolution graphics

Type of graphics for the APPLE II which has a 40x40 graphics screen and four lines of text, 15 colors and certain commands to plot squares (PLOT) and draw lines (HLIN, VLIN). There is another graphics mode, high resolution, which will be covered in the next lesson. The command GR places the APPLE II in low-resolution graphics.

nested loops

Loops which fit inside each other. (i.e. in one cycle through the outside loop, the inside loop may be completed many times.

charting a program

A simple system for following each step of a sequence of instructions. Charting is especially helpful to understand the changing values during a loop. The process is sometimes called "walking through a program."

D. Commands

```
FOR A=1 to 10
....
NEXT A
```

Command telling computer to set the value of A at 1, to complete whatever commands are listed in the next lines and at NEXT A to return to For, set the value of A at 2, etc. When the value of A is becomes 10, the loop is on its last cycle and the computer will continue to the following statement.

An empty loop (no command between the For and Next commands) can be used to slow down the processing of a program. This is especially helpful in graphics programs where the commands may be carried out too fast for our poor eyes.

The computer insists that each For statment have a NEXT associated with it and will respond with an error message if you should leave it out.

STEP +2

A loop is normally incremented (changed) by adding one (A=1 the first time, A=2 the second), however, other units can be used (both plus and minus.)

GR

Sets the computer to low resolution graphics

COLOR=

Sets the color of the line to one of 15 available colors. If not specified, computer will set the value to 0 (black).

PLOT A,B

Draws a box on the screen in column #A (0 of left, 39 on right) in row B (0 at the top, 39 at the bottom).

HLIN C,D at E.

Draws a line from column #C to column #D at row #E

VLIN F,G at H

Draws a line from row #F to row #G at column #H.

TEXT

Returns the computer to the text (writing) mode so that the entire screen is available for writing programs or for use with the PRINT or

INPUT command.

E. Assignment

1. Rewrite 1 potatoe,2 potatoe using loops.

2. Design some lovely shape using Low Resolution graphics paper. Write a program to put your design on the screen. (If you have a color monitor, take advantage of the color options.)

3. Write a program to move a little bug around the screen. To move a bug one space, first plot the square, then put in a pause loop, then erase the bug by plotting it in black (color=0). Remember to reset the color for the next square; the movement can be done inside a loop.

4. It is not possible to put text on the graphics screen, however, you can label the elegant design you made in #2 by adding these lines:

```
2000 HOME
2010 PRINT TAB(7)"THIS IS A BEAUTIFUL DESIGN"
2020 PRINT
2030 PRINT TAB(3)"IT WAS MADE USING LOW-RES GRAPHICS"
2040 GOTO 2040
```

(Line 2000 puts the cursor at the top left of the text area; line 2040 produces an infinite loop to keep the cursor off the screen. To stop this program, use CTRL/C.)

Quiz on lesson 4

loop

infinite loop

nested loops

graphics mode

text mode

low resolution graphics

charting a program

FOR D=1 TO 3

NEXT D

FOR Z= 6 to 2 STEP -2

NEXT Z

GR

TEXT

COLOR =

PLOT 10,36

HLIN 4,25 AT 0

VLIN 0,5 AT 4

BASIC

LESSON FIVE: RANDOM NUMBERS AND HIGH RESOLUTION GRAPHICS

A. Random Numbers

Random numbers are defined as numbers which are not in a predictable sequence. Deciding the order of correct answers for a quiz, winning numbers for a lottery, calling heads/tails are everyday uses for random numbers. Those of you who have used random numbers to select a sample for a research project probably remember the random number tables in the back of the statistics book. In computer programming random numbers can be used in many simple-minded and effective ways.

1. In order to see what the computer produces when you ask for a random number, type in the following program.

```
NEW

10 PRINT RND(1)

20 GO TO 10

30 END
```

Try it. Fine, but the numbers are all decimals between 0 and 1. Let's imagine that you want random numbers between 1 and 15 (which happens to be the colors available in graphics). In order to get the numbers to come within that range you will need to multiply them all by 16 (since none of the numbers is exactly 1, none of the new numbers will be 16). So change line 10 to read:

```
10 PRINT 16*RND(1)
```

Try it. Fine, but what about all those decimal places. Aha, there is a command in BASIC that will turn those nasty numbers into whole numbers, called integers (INT). Make this change

```
10 PRINT INT(16*RND(1))
```

The double parentheses are needed because INT requires that whatever is going to be made into an integer appear inside parentheses.

Try it. Run it a few times to see that the sequence is indeed different.

2. Let's use it in a program:

```
NEW
```

```
10 REM PROGRAM TO DEMONSTRATE USE OF RANDOM NUMBER GENERATOR
20 GR
30 FOR A = 1 TO 39
50 COLOR=INT(16*RND(1))
60 HLIN 1,39 AT A
70 VLIN 1,39 AT A
80 NEXT A
100 END
```

Try it. Run it a few times, to see the different patterns that will be generated. You can add a time lapse loop, all kinds of things.

3. If you wanted to generate a random list of correct answers for a true and false test, you would only need to multiply the random number by 3 (1=false, 2=true). Write a loop producing such a list for 10 true/false questions. Run it a few times to make sure it really is random.

B. High Resolution Graphics.

There is a second type of graphics available on the APPLE II with different commands

HGR draws in high resolution where the grid is 280 points across and 160 points down: i.e. much finer and more delicate drawings can be produced. There are still text lines at the bottom.

HCOLOR= Unfortunately the trade-off with more points is less colors. There are only 7. They will vary between brands of monitors, there may even be differences in color between lines drawn in even numbered columns and odd numbered columns (honest):

0= black
1= green (depends of monitor)
2= blue (depends on monitor)
3= white (2)
4= black (2)
5= depends on monitor
6= depends on monitor
7= white

You obviously need to make a correct list for your machine.

HPLOT X,Y (same as in low resolution PLOT, but there are more potential positions. 0,0 remains in upper left corner.

HPLOT X,Y TO W,Z functions like the HLIN and VLIN except that any two points can be connected (or using a formula a curve can be plotted). These can be in a chain HPLOT A,B TO C,D TO E,F TO G,H

2. Let's change the random color program above to high resolution graphics. You can retype individual lines or just write a new program.

```
NEW

10 REM PROGRAM TO DEMONSTRATE HIGH RESOLUTION GRAPHICS

20 HGR

30 FOR A=1 TO 160

40 HCOLOR =INT(8*RND(1))

50 HPLOT 0,A TO 279,A:REM DRAWS HORIZONTAL LINE

60 HPLOT A,0 TO A,160 :REM DRAWS VERTICAL LINE

70 NEXT A

100 END
```

Try it. Notice the difference between high and low resolution in the colors, width of lines and speed. I find that, in order to use HGR effectively, I have to keep a piece of graph paper labelled with the coordinates at my side. Spend some time trying different shapes, designs.

By the way, you can get different background colors in hi-res graphics by calling a machine language subroutine: Set HCOLOR to the background color you want, plot anything (it won't show anyway) and then CALL 62454.

```
10 HGR

20 HCOLOR=5

30 HPLOT 0,1 TO 1,2

40 CALL 62454
```

50 (now go on to set a new color and draw lines on your background)

If you have a dot matrix printer and an interface card that allows you to do graphics dumps, you can print a high resolution graphics screen on your printer. (You cannot print low resolution graphics.) Check the directions in the manual for the interface card and the manual for the printer. If you have an Epson Printer and a Grappler Interface card, get the high resolution graphics picture on the screen, turn on the printer, make sure the paper is properly aligned, then type:

```
PR#1

PRINT CHR$(9);"G"
```

and the printer should begin reproducing the screen. For additional commands, check the manuals, or the HEC Manual #11 USING YOUR EPSON PRINTER, p.14.

E. Terms

random numbers

Numbers which do not appear in a predictable sequence.

high resolution graphics

More points available of the screen than low resolution graphics, but less colors. Different commands are used for high and low res graphics.

F. Commands

RND(1)

Produces a decimal random number between 0 and 1

INT(RND(1))

Produces a random whole number between 0 and 1

INT(6*RND(1))

Produces a random whole number between 1 and 5

HGR

Command to use high resolution graphics

HCOLOR

Command to set color in high resolution graphics. There are 7 color choices with 0=black and 7=white. Others vary with brand of monitor

HPLOT X,Y

Places a dot on screen at the intersection of column X and row Y. There are 279 columns and 160 rows.

HPLOT X,Y, to W,Z

Places a line from the intersection of column X row Y to the intersection of column W row Z

F. Assignment

1. Using RND to generate the number, write a guessing game in which the user if told if his guess is too high or too low.

2. Add to Assignment #1 to produce some classy high resolution graphics when the user guesses the number correctly.

BASIC

LESSON SIX: SUBROUTINES AND THINGS THAT GO PEEP IN THE NIGHT

A. Subroutines

The more complex your programs become, the more confusing they will get for you, the program reader. While there is a certain diabolic joy in writing a program so complex it defies anyone to follow it, Good Little Programmers to not set obfuscation as their goal in life. If you are teaching programming, it is legitimate to reward students for both documentation (REM statements, diagrams) and for clear logic. Not only does this enhance the sharing of programming solutions between students, it prepares them for the vocational expectation of clear programming. Traditionally the person called in to change a program to meet new user needs is not the original programer. Even if it is the original programmer, that person will certainly not remember the logic used 200 programs ago. The best punishment for a sloppy program is probably making the programmer make a minor change. You can make up a new verse to that wonderful Mikado patter song about letting the punishment fit the crime.

(An aside to the aside. Clear logic does not mean Your logic. Perhaps the greatest joy of programming is that there is ALWAYS another way to do ANYTHING. Students should be encouraged to try a variety a ways to achieve the same (or similar) results.)

There are two other reasons to use subroutines: (1) if you have repetitive commands, you only have to type them in once and (2) you can test the workings of the subroutine by typing in the values for any variables you may need and then GOSUB 1000 (and the computer will RUN only that piece.)

1. Which brings us to subroutines.

Type in the following program

```
NEW
10 REM PROGRAM TO DEMONSTRATE SUBROUTINES
20 GOSUB 1000: REM TEXT SUBROUTINE
30 GOSUB 2000: REM LOW RES GRAPHICS SUBROUTINE
50 END

1000  REM SUBROUTINE TO PRINT OUT EXPLANATION OF PROGRAM
```

```
1010 PRINT "THIS PROGRAM WILL DEMONSTRATE 5 GOSUB ROUTINES
1020 GOSUB 4000:REM LOOP TO KEEP PRINT STATEMENT ON THE SCREEN
1100 RETURN


2000 REM LOW RESOLUTION GRAPHICS PROGRAM
2010 GR
2020 FOR A=1 TO 39
2030 COLOR =INT (16*RND(1))
2040 HLIN 1,39 AT A
2050 VLIN 1,39 AT A
2060 GOSUB 4000: REM PAUSE TO SLOW DOWN DRAWING SO IT CAN BE
ENJOYED
2070 NEXT A
2080 RETURN


4000 REM SUBROUTINE TO CREATE PAUSE
4010 FOR PAUSE =1 TO 1000
4020 NEXT PAUSE
4030 RETURN
```

Neat isn't it. Note that the RETURN command is required. (After all, think what would happen if it weren't included) and that the RETURN brings the computer back to the end of the GOSUB line, ready for the next line.

Note that the remark (REM) statement can also be used as a second command on one line (after a :). The advantage is obviously that it saves space; the disadvantage is that a mistake takes more time to retype.

GOSUB also can be used in conditional branching statements:

```
If SUM=2 THEN GOSUB 1000
```

B. SOUNDS

Now you too can sound like an arcade. Actually some small, inoffensive sounds can be very helpful in programming. However, my experience is that when an entire class is typing in the following exercise, everyone gets edgy. Be careful how much electronic noise you introduce into your (or the students') environment.

In order to hear the different sounds available to you, you will need to write a machine language subroutine (wait until you tell that to the folks at home).

1. Type

```
NEW

10 REM MACHINE LANGUAGE SUBROUTINE TO PRODUCE SOUNDS

15 REM LINES 20-60 ESSENTIAL FOR ANY SOUND-PRODUCING PROGRAM

20 FOR I=770 TO 790

30 READ J

40 POKE I,J

50 NEXT I

60 DATA
173,48,192,136,208,5,206,1,3,240,9,202,208,245,174,0,3,76,2,3,96

100 REM COMMAND TO SET SHORT DURATION FOR SOUND

110 REM VALUE IN 769 CAN BE ANY NUMBER FROM 1 TO 255

120 POKE 769,5

200 REM LOOP TO MAKE ALL 255 SOUNDS

210 FOR S=1 TO 255

220 POKE 768,S

240 REM NEXT COMMAND CALLS SUBROUTINE, NOTHING WILL HAPPEN
WITHOUT THIS COMMAND

250 CALL 770

260 NEXT S
```

Run this program. There are, of course, an infinite number of changes. You can run only some of the notes. (Change the value of the loop in line 210. Remember that you can use STEP with positive and negative numbers.) Add a graphics program so that, as the vertical and horizontal lines are drawn, there are little peeps. Or write a little quiz program (INPUT A and B, what is the SUM, IF SUM <> A+B ask again, IF SUM=A+B play a tune. <> means "not equal to", it can also be written ><

2. By the way, PEEK ## looks into a memory location of the computer. POKE ##,xx looks into a memory location (##) AND place a value into it (xx). It is always safe to PEEK; it is definitely not always safe to POKE; you might wipe out your program. (You cannot do the computer permanent damage however. Once you turn the computer off your POKES are gone. The computer won't let you POKE into its permanent memory (which is called READ ONLY MEMORY or (ROM) i.e. you can read what is in there (PEEK), but you can't change it (POKE). See how good you are getting at Cocktail Computertalk?

3. The CALL command sends the computer to the address specified to find a subroutine, execute it and, finally, return to the next line of the program. In other words, CALL in machine language acts line GOSUB/RETURN in BASIC.

If you want to start composing, you can use these pitch equivalents developed by South Hadley High School students:

| Value | Note | Value | Note |
|---|---|---|---|
| 16 | C | 66 | C |
| 17 | D | 70 | D |
| 20 | E | 80 | E |
| 22 | F | 90 | F |
| 25 | G | 100 | G |
| 26 | A | 106 | A |
| 29 | B | 118 | B |
| 33 | C | 132 | C |
| 35 | D | 140 | D |
| 40 | E | 160 | E |
| 45 | F | 180 | F |
| 50 | G | 200 | G |
| 53 | A | 212 | A |
| 59 | B | 236 | B |

C. On...(a special form of conditional branching)

Imagine that you gave the user a choice between selecting the numbers 1, 2, 3, and 4. As soon as you get the number the user selects, you want to branch to different responses. You could always write:

```
NEW

10   IF A=1 THEN GOTO 100
```

```
20   If A=2 THEN GOTO 200

30   IF A=3 THEN GOTO 300

40   If A=4 THEN GOTO 400

50   GOTO 5:REM re-ask the question if input not 1 to 4.
```

There is a shorthand way, as long as the choices are sequential beginning with 1.

```
10 ON A GOTO 100,200,300,400

20 GOTO 5
```

These instructions mean: If A=1 then go to line 100; if A=2 then go to line 200; if A=3 then go to 300; if A=4 then go to 400; if A doesn't equal any of these, go to the next line (20). Neat isn't it. You can list 270 choices (i.e. A can equal everything up to 270 which seems slightly improbable, but anyway...)

This command is frequently used for "menu selection" where the user is given a choice of activities, or for any multiple choice activities.

E. Terms

subroutines

A programming techniques that enables to programmer to separate the total program into descrete sections. Very useful for repetative sections of a program. Subroutines always return to exactly the same line from which they started.

F. Commands

55 GOSUB 400

Sends the computer to line 400. On the RETURN line, computer will return to the end of line 55.

Places the value yy into a memory location xx. Used to produce effects not directly available through BASIC.

CALL xx

Command to call a machine language subroutine stored beginning at address xx.

If A<> 4 THEN...

Conditional branch. If A does not equal 4 the rest of the command will be carried out.

If A$="YES" THEN PRINT "YOU ARE RIGHT":GOTO 10

Conditional branch. If A$ does equal "yes" then the computer will print out "YOU ARE RIGHT" and will go to line 10. If A$ does not equal yes, nothing will be printed AND the computer will not go to 10 (it will go to the next line).

ON A GOTO 100,150,200 (or ON A GOSUB 100, 200,300,400)

If the value of A is 1 go to line 100, if the value of A is 2 go to line 150, if the value of A is 3 go to line 200. Very useful branching statement. If the value of A is greater than the list, computer goes to the next line.

F. Assignment

1. Look back over programs you have written, which ones would be improved by subroutines. Rewrite that program. (If you can't find one that is beggi to be recycled, write a new one.)

2. Complete, or write a different version of, the following program:

```
10 FOR C=1 TO 2
20 HOME
30 PRINT "ON THE ";
40 ON C GOTO 200,210
200 PRINT "FIRST ";:GOTO 500
210 PRINT "SECOND ";:GOTO 500
500 PRINT "DAY OF CHRISTMAS"
```

```
510 PRINT
520 PRINT "MY TRUE LOVE GAVE TO ME:"
530 PRINT
540 ON C GOTO 1000,990
990 PRINT "TWO TURTLE DOVES AND"
1000 PRINT "AND A PARTRIDGE IN A PEAR TREE"
1010 FOR P=1 TO 2000:NEXT P
1020 NEXT C
```

Quiz Lesson 6

subroutines

GOSUB 250

RETURN

ON D GOTO 350,750,1050,1550,2050

POKE 768,15

CALL 770

BASIC

LESSON SEVEN: 2-DIMENSIONAL ARRAYS, FOOLING AROUND WITH STRING VARIABLES

a. 2-Dimensional Arrays (or Tables)

1. Remember the problems we had earlier trying to develop a list of student records? Reading in one student name and one test score was fine, but what should the second score be called? And the third test? When the second student's name was read, it replaced the first student's name. Well, now you have the skills to do the fancy programming that datebased management programs use to keep track of information. You just make creative use of those loops.

Now, instead of the variable named student, we want a variable which is student (1), student (2), student(3) and so on. Instead of a test score for student one, and then another score, we want test scroe (student one, test #1), score (student one, test #2), score (student one, test #3) and so on.

What we want, ahd what we have, is subscripted variables. (Of course, a true subscript goes below the line, but this is not possible on the computer, so the below-the-line numbers are put in parentheses.)

Type in this program

```
NEW

10 REM PROGRAM TO DEMONSTRATE TWO-DIMENSIONAL ARRAYS

15 PRINT "NAME" TAB(15) "GRADES" TAB(35) "AVG."

20 REM

       OUTER LOOP FOR COLUMN OF STUDENT NAMES

30 FOR R=1 TO 3

40 READ S$(R):REM S$(R)=STUDENT NAMES

50 PRINT S$(R);TAB(15);:PRINTS FIRST PART OF LINE

60 REM

      INNER LOOP TO READ TEST GRADES

70 FOR C=1 TO 5

80 READ G(R,C)

90 REM

      SUM TO CALCULATE AVERAGE
```

```
100 T(R)=T(R)+G(R,C)

110 PRINT G(R,C);"  ";: REM PRINTS SCORES

120 NEXT C

130 PRINT T(R)/5: REM PRINTS AVERAGE

140 PRINT

150 NEXT R

500 DATA JOAN SMITH, 50,60,50,60,50

510 DATA IAN JONES, 70,80,70,80,70

520 DATA ANN DOE, 80,90,80,90,80

1000 END
```

After you run this program, create one of those famous charts to make sure that you understand what has happened.

Another technique in debugging a program is to add the STOP command at the end of some part of the program. The program will run to that point. When you are sure that everything is happening correctly, type CONTINUE (and it will). You can put a number of STOP lines in a program, taking them out when you are through.

2. You can expand this program to take input (change the read statements to input, take out the data statements). You can change the number of students from a set number of three to a variable (INPUT "HOW MANY STUDENTS WILL YOU ENTER";SN and then the loop would read FOR R=1 TO SN). You can make the same change in the number of test scores or grades (input to find out number of grades, change loop to a variable number and change the average (line 130) from 5 to the same variable)

3. When the Apple comes across a variable with one, two (or more) dimensions, it automatically saves 10 boxes for each dimension i.e. the highest value of G(R,C) is G(10,10), or a table with 10 names each way. If you plan to have more than 10 rows or columns, you will need to add a statement AT THE BEGINNING of the program telling the APPLE to save more boxes.

```
7 DIM G(50,50)
```

or whatever.

Once you have established a variable as being an array (table), you must always refer to it that way. Remember that under normal circumstances, the computer is often told to assign a value ("MARY" for example) to a variable (S$) and later in the same program to assign S$ a different value ("JOHN"). With dimensioned variables, the computer is told S$ is a series of boxes, each with a different value. And you, dear computer, had better keep them all straight. The least you can do, dear programmer, is to keep the name of the array (table) the same throughout the program.

B. String Variable Manipulations

1. Since a string variable is read into the computer character by character, the computer can be asked to produce any character you are interested in. You, as programmer, can attack your string from the left or from the right:

LEFT$(F$,1) tells the computer to start on the left-hand side of the string labeled F$ and to get the first character. If the string variable were dimensioned, it would read LEFT$(N$(R,C),1)

RIGHT$(F$,1) is exactly the same, but produces the right-hand-most (or last) character.

MID$(F$,1,1) will produce the same letter as the left above, but works slightly differently. It tells the computer to begin at the left of the variable labelled F$, count in to the first number (in this case 1), print the number of characters indicated in the next number (in this case one). In MID$(F$,2,3) when F$="MARY," the command would get ARY.

2. Try the following routines

```
NEW

10 REM PROGRAM TO DEMONSTRATE LEFT, RIGHT AND MID STRINGS

20 A$="GOOD MORNING WORLD"


30 FOR P=1 TO 18

40 PRINT LEFT$(A$,P)

50 NEXT P

60 END
```

Try it, then change

```
 40 PRINT RIGHT$(A$,P)
```

And then

```
40 PRINT MID$(A$,P,2)
```

There is one more change, make the computer count the letters for you:

```
30 FOR P=1 TO LEN(A$)
```

And of course you can now make the program more flexible by putting in an INPUT statement

```
20 INPUT "TYPE IN YOUR NAME ";A$
```

And if you want to make it into a repeating pattern, you can either add a large outside loop (at 25 and 55 or just a GOTO 30 at line 55.

3. As long as you are playing with string variables, they can also be added:

```
NEW

10 PROGRAM TO DEMONSTRATE CONCANTATION

20 LET T$="TEACH"

30 LET I$="ING"

40 PRINT T$+I$
```

C. ASCII (American Standard Code for Information Interchange)

As you already know, if you tell the computer that the value about to be entered is a number and you type in a letter, the computer has a fit. There really is a good, electronic reason for the problem. Each character on the keyboard, each character produced by pressing the shift and a key simultaneously and each character produced by pressing the CTRL and a key simultaneously are stored as separate binary values in the computer. (Binary is a system for counting where the only digits avaialable are 0 and 1). All the computer really uses in its electronic soul is binary numbers (0=the little micro-switch if off (low current);1=the little micro-switch is on (high current). That statement always reminds me of the Moment of Truth when Dorothy and her Friends learn who the Wizard really is.)

These binary values are first translated into hexidecimal values (base 16) and then to decimal values (base 10 remember ?) We will deal with these things in the next lesson in more detail.

It it kind of interesting to find out the different decimal values of the letters and numbers you have typed.

Type

```
PRINT ASC("A")
```

and so on,

If you want to check the system asked the computer to

```
PRINT CHR$(65)
```

A list of the ASCII code values can be found in the back of most computer and BASIC manuals.

There will be situations where you can actually use these commands, but for the time being, I think it helps explain why numeric and string variables are different. By the way, if you have some data which has been collected as a numeric variable (like year of graduation) which you want to use as a string variable (for example printing out only the last two characters) you can convert variable type.

```
NEW

10 LET Y=1984

20 LET Y$=STR$(Y)

30 PRINT RIGHT$(Y$,2)
```

or, in reverse

```
10 LET Y$="1984"

20 LET Y=VAL(Y$)

30 PRINT Y+2
```

D. The GET command

1.You have already learned how to make the program pause. If you want to stop the program until the user is ready to continue, write

```
100 INPUT "PRESS ANY KEY AND THE RETURN KEY TO CONTINUE";C$
```

The computer will accept any alphanumeric value and will go on. If you had used C instead of C$ what would happen?

2. Or you can remove the need for the RETURN key by using GET instead of INPUT. GET has the disadvantage of accepting only one character (but the advantage on not requiring a RETURN.)

```
100 PRINT "PRESS ANY KEY TO CONTINUE"
110 GET A$
```

GET is very convenient in minor commands "DO YOU WANT TO CONTINUE?

Y/N"

3. You can also use GET in this need little, and possibly useless, program which I love:

```
NEW

10 PRINT "TYPE IN ANYTHING YOU LIKE"
20 PRINT:PRINT:PRINT

30 GET B$

40 PRINT B$;

50 GOTO 30
```

To stop this program you have to hit RESET (or CTRL/RESET). If that offends you, fugure out how to introduce an IF statement and come uncommon key to stop it. If you want everything printed on the printer add

```
5 PR#1
```

4. George Smith uses a wonderful little graphics program using GET

```
NEW

10 GR

30 INPUT "TYPE IN STARTING POSITION X,Y ";X,Y

40 COLOR=15

50 PLOT X,Y

60 PRINT "MAKE A MOVE U-D-L-R"

70 GET A$

80 COLOR=0:REM BLACK WILL "ERASE SQUARE"

90 PLOT X,Y

100 IF A$="U" THEN 200

110 IF A$="D" THEN 300

120 IF A$="L" THEN 400

130 IF A$="R" THEN 500

140 GO TO 600

200 Y=Y-1:GOTO 40
```

```
300 Y=Y+1:GOTO 40

400 X=X-1:GOTO 40

500 X=X+1:GOTO 40

600 PRINT "ERROR-TRY AGAIN":GOTO 60

1000 END
```

When we were using the program in class one night, Larry Boucher made a wonderful change in it to turn the bug into a LOGO like character. He changed line 80 to COLOR = 2. Try it. Make sure you understand what is happening. (After that the variations were infinite, different colors for different directions, changing colors randomly etc.)

E. TERMS

Array

A list or table (a two-dimensional array) in which the addresses of the values are specified with subscripted numbers. For example in the array G(C) where G= grades and C=the 10 grades, the boxes are labelled as follows:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|

in the array G(R,C), where R=3 students the boxes are labelled as follows:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 | 1,7 | 1,8 | 1,9 | 1,10 |
| 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | 2,6 | 2,7 | 2,8 | 2,9 | 2,10 |
| 3,1 | 3,2 | 3,3 | 3,4 | 3,5 | 3,6 | 3,7 | 3,8 | 3,9 | 3,10 |

arrays make the collecting and using of data infinitely easier than when every variable has to be named separately.

concantation

the addition of two or more string variables

ASCII

American Standard Code of Information Interchange. The Decimal value of each keyboard character (straight keyboard, shift/keyboard, CTRL/keyboard). All microcomputers use the same code (that's why it is "standard".

binary

Base two number system used by the computer since the computer only has the option of having its switches off(0) or on(1). Each character typed into the computer is translated to a binary code.

E. Commands

DIM S$(25,20)

Saves space for a table 25 rows by 20 columns. If you do not dimension a variable Apple automatically save 10 spaces in each dimension.

S$(A,B)

Table, or array of string variables labelled S$ which has A number of rows and B number of columns. S(A,B) would be an array of the same size of numeric variables.

PRINT LEFT$(V$,2)

Command to print the first two characters in the string variable V$

PRINT RIGHT$(V$,2)

Command to print the last two characters in the string variable V$

PRINT MID$(V$,3,2)

Command to print the third and fourth characters in the string variable V$

PRINT LEN(V$)

Command to print the number of characters in the string variable V$

PRINT ASC("S")

Command to print out the decimal value in the American Standard Code (called "asskey code") for the character S

PRINT CHR$(126)

Command to print out the character stored as ASCII code 126

LET Y$=STR$(Y)

Command to convert the numeric variable Y into a string variable Y$

LET Y=VAL(Y$)

Command to convert the string variable Y$ into a numeric variable Y

INPUT "TO CONTINUE, PRESS ANY KEY AND THE RETURN";A$

Command to stop processing of a program until the user presses the keyboard. Used especially to slow down printing so user can read directions, make decisions.

PRINT "PRESS ANY KEY TO CONTINUE"
GET Z$

Does not require the RETURN key. Will accept only one character (unlike input which can take a string or a numberic variable).

STOP

Command placed temporarily in a program for debugging purposes. To go on, type CONTINUE.

F. Assignment

1. Write a simple (3 students, 4 scores) program which reads in names and grades and prints out name and average. (You will only need a one-dimensional array).

2. Write a simple mailing list program, which uses an input statement to take 3 people's first and last name, address (street,town,state,zip) and two code numbers. You will need a two-dimensional array.

Print out the list so that you have only the information necessary for mailing labels.

Print out a second list that has last name, first initial and one code number.

Finally, make up a brief form letter, insert the first name, with the proper spacing, into each copy of the letter.

QUIZ ON LESSON SEVEN

array

binary

DIM

LEFT$(A$,X)

RIGHT$(A$,Y)

MID$(A$,M,N)

LEN(A$)

STOP

STR$(A)

VAL(A$)

ASC("B")

CHR$(66)

GET T$

BASIC

LESSON EIGHT: FILES, HEXIDECIMALS, AND FLOWCHARTING

A. Files

1. There are only a limited number of ways computers can get the data (information) they need in order to carry out the programs people write for them:

a. they can be given it directly by the user (the old INPUT statement),

b. they can read it from the program itself (as in READ/DATA),

c. they can generate it themselves by

(1) iteration (N=N+1, or FOR I = 1 to 10/NEXT I, or

(2)' by performing some operation (LET X=3*2/A, or LET N$=V$+LEFT$(Z$,3),

d. they can read it from a file where information has previously been stored.

Obviously we are going to deal with the last of these, since you know how to do all the others.

There are two kinds of files:

Sequential Access Files: Each record uses just as many character spaces as are typed in, next record starts in the next space. Very good for not using any extra memory, very bad for ordinary, everyday people. All THE BOOKS say that sequential files are best used when the file doesn't have to be changed. If you have information on a list that doesn't often have to be changed, use a tyepwriter, or chip it is stone; don't waste computer space. The only analogy I can think of is having a file draw of folders, say one per student, but not having anything written on the tabs, so you have to pull out each file in turn until you find the one you want. I don't like sequential files.

Random Access Files: You tell the computer how many bytes per record you want the computer to save. If you want 40 bytes saved, it will do so. The computer can then easily find student record number 5 because it just counts along from 1 to 41 to 81 etc. Now this may waste space (especially if you only have used about 15 bytes per student) so you may have to break up long files into two disks, but

life is much easier. This lesson will only include directions for using random access files. (By the way, each character (or space) takes one byte, each number takes 5 bytes.)

2. To make files slightly less burdensome, remember that the statement PRINT is used by the computer as having a wider meaning than the Guttenberg Bible. PRINT means "do something with this stuff", "put it somewhere," "output it." Life would probably be easier it we used the word OUTPUT instead of PRINT, but we don't.

In order to let the computer know that you want make contact with a file which is stored on a disk, you use a secret code created by simultaneously pressing the CRTL and the D keys. Now, as you know CTRL/key combinations don't show on the screen, which makes it hard to remember if you really pressed the ctrl/d or not. BUT, as you also know, every control/key combination has its own ASCII decimal code. The ACSII code for CTRL/D is 4.

So the first thing that happens in any program using a file is to establish a variable which lets the computer know that the CTRL/D is being pressed. This is usually written as

```
10 DOS$=CHR$(4)
```

OR

```
10 D$=CHR$(4)
```

Then, in the program, whenever you want the CTRL/D to be pressed, you write PRINT DOS$ (or PRINT D$).

Obviously any string variable name can be used, but using the same one all the time, keeps you from using the same variable name during the program for some other value.

To review: everytime the computer receives the message that CTRL/D has been pressed (by having a command in the program saying output (PRINT) the ASCII value of CTRL/D, CHR$(4), the computer will expect the command to refer to a file.

3. There are really only five commands associated with using random files. Notice that the syntax (;" ") remains the same.

```
DOS$=CHR$(4)

PRINT DOS$;"OPEN DEMO1,L150"
```

Line 100 tells the computer to open an already established file named DEMO1 in which each record has 150 bytes. If there is no such file, this command creates the file.

```
PRINT DOS$;"WRITE DEMO1,R";X
```

Tells the computer to put the next information (which will be in PRINT

statements) into the random access file called DEMO1. The X at the end indicates that this will be record # (whatever value you have given x). The command which follows is to print (or put) into the file and is, therefore, PRINT A$,B,C (or whatever values you want printed into the record). You must make sure that you do not put in values that are greater than the number of spaces you saved (remember the L150?). If you do, the computer will begin writing in the next file and you are doomed with a horrible mess you can never untangle.

```
PRINT DOS$;"READ DEMO1,R";X
```

Tells the computer to read record number (whatever value you have given X) from the random access file named DEMO1. The command which follows takes input from the file and is, therefore, INPUT A$,B,C (or whatever the values you have stored)

```
PRINT DOS$;"CLOSE DEMO1"
```

Tells the computer to close the files. If that is not done EVERYTHING is lost. Close really means "save". A Good Programmer does not open the file in a program until absolutely necessary and closes it again as soon as possible. Not only does this avoid doing something unfortunate during the program, but it minimizes the risk from suicidal squirrels, lightening storms, and bells that tell you that you are late for class.

```
PRINT DOS$
```

Used by itself this command tells the computer to cancel whatever it has been doing with the file, i.e. if it has been reading from the file, and you now want it to read from the keyboard, you would use this. This command does not have the ability to save what it has done, only CLOSE does that.

4. Now let's try it. Load your own initialized disk so that there will be some place to save the file.

Type in the following program:

```
NEW

10 REM PROGRAM TO DEMONSTRATE STARTING A RANDOM FILE

20 DOS$=CHR$(4)

30 INPUT "HOW MANY STUDENTS? ";I

40 DIM N$(I),G1(I),G2(I): REM DIMENSIONED IN CASE MORE THAN 10
RECORDS

50 FOR R=1 TO I
```

```
 60 PRINT "STUDENT #";R;
 70 INPUT N$(R)
 80 INPUT "FIRST SCORE ";G1(R)
 90 INPUT "SECOND SCORE ";G2(R)
100 PRINT
110 NEXT R
120 PRINT DOS$;"OPEN GRADES,L150"
130 FOR Q=1 TO I
140 PRINT DOS$;"WRITE GRADES,R";Q
150 PRINT N$(Q)
160 PRINT G1(Q)
170 PRINT G2(Q)
180 NEXT Q
190 PRINT DOS$;"CLOSE GRADES"
200 PRINT "THE INFORMATION IS IN THE FILE"
210 END
```

Great. Now you have all this stuff in, can you get it out? If you catalog your disk you will find that Grades is listed as a Text File (T in the first column). If you ever want to see that data again, you will have to access it through a program designed to read the files. (A little imagination will convince you that it is important to have written down what you files look like, or you will not know how to get them out.)

Save the program above RIGHT NOW (before you type NEW), or you will not be able to use this program later.

```
NEW

10 REM PROGRAM TO DEMONSTRATE READING A RANDOM ACCESS FILE

20 DOS$=CHR$(4): REM SET VALUE FOR CONTROL/D

30 INPUT "HOW MANY STUDENTS? ";I: REM NEEDS TO MATCH OR BE LESS THAN
THE NUMBER ENTERED EARLIER

40 DIM N$(I),G1(I),G2(I):REM IN CASE THERE ARE MORE THAN 10 IN THE
FILE
```

```
50 PRINT DOS$;"OPEN GRADES,L150"

60 REM
        LOOP TO READ OUT NAMES,GRADES
70 FOR Q=1 TO I
80 PRINT DOS$;"READ GRADES,R";Q
90 INPUT N$(Q),G1(Q),G2(Q)
100 NEXT Q

110 REM
        CLOSE FILES
120 PRINT DOS$;"CLOSE GRADES"
130 PRINT DOS$: REM RETURNS TO NON-FILE COMMANDS
140 REM
        LOOP TO PRINT FILES
150 FOR Q=1 TO I
160 PRINT N$(Q);TAB(15);G1(Q);"  ";G2(Q)
170 PRINT
180 NEXT Q
200 END
```

5. Files are both essential and tricky. I would strongly recommend developing file programs in bits and pieces, using only a few records with a few bits of information. (This is exactly the place for subroutines). Trying do too much without checking each piece produces something akin to a kitten with a ball of yarn.

6. TRACE cannot be used in programs which call disk files (can't write to files when TRACE is on.) Instead use MON C,I,O (Monitor: commands, input, output). When you are through type NOMON C,I,O

B. HEXIDECIMAL BINARY: The only way I know to explain hexidemicals is to use them. As you may remember, the computer, at it's most fundemental level, uses electric circuits which are either on or off (actually high and low current). The technological revolution which

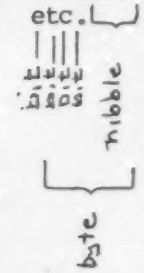
made microcomputers possible was the miniturization of these electronic circuits.) Therefore, at the electronic level the computer can be said to be using a binary system (0 and 1).

To describe the on/off configurations is a more reasonable way that just listing 01001010, a second level system has been developed. Since the on/off circuits are divided into groups of 8 (subdivided into 2 groups of 4), AND since it was important to use as few spaces as possible to describe the pattern, hexidecimal was used. The only way to understand why it to do some counting.

But before we count, the patterns of on/off are also translated into the decimal equivalent for the convenience of the programmer. (If you are using this chart alone go line by line holding a paper over the next line so that you can try to anticipate what is coming next. When you are finished, make a blank chart and fill it in by yourself.)

| BINARY | HEXIDECIMAL | DECIMAL |
|---|---|---|
| 00000000 | 00 | 0 |
| 00000001 | 01 | 1 |
| 00000010 | 02 | 2 |
| 00000011 | 03 | 3 |
| 00000100 | 04 | 4 |
| 00000101 | 05 | 5 |
| 00000110 | 06 | 6 |
| 00000111 | 07 | 7 |
| 00001000 | 08 | 8 |
| 00001001 | 09 | 9 |
| 00001010 | 0A | 10 |
| 00001011 | 0B | 11 |
| 00001100 | 0C | 12 |
| 00001101 | 0D | 13 |
| 00001110 | 0E | 14 |
| 00001111 | 0F | 15 |
| 00010000 | 10 | 16 |
| 00010001 | 11 | 17 |
| 00010010 | 12 | 18 |
| 00010011 | 13 | 19 |
| 00010100 | 14 | 20 |
| 00010101 | 15 | 21 |
| 00010110 | 16 | 22 |
| 00010111 | 17 | 23 |
| 00011000 | 18 | 24 |
| 00011001 | 19 | 25 |
| 00011010 | 1A | 26 |
| 00011011 | 1B | 27 |
| 00011100 | 1C | 28 |
| 00011101 | 1D | 29 |
| 00011110 | 1E | 30 |
| 00011111 | 1F | 31 |
| 00100000 | 20 | 32 |
| 00100001 | 21 | 33 |
| 00100010 | 22 | 34 |
| 00100011 | 23 | 35 |
| 00100100 | 24 | 36 |
| 00100101 | 25 | 37 |
| 00100110 | 26 | 38 |
| 00100111 | 27 | 39 |
| 00101000 | 28 | 40 |

etc.



You will certainly never need to memorize the chart you have just made. We have gone through it so that you will understand why people

go around talking about HEX and BINARY. More important, the groups you just used are BYTES and BITS and NIBBLES, terms which presumably computer people took from Weight Watchers.

Byte=the group of 8 binary numbers (or 8 switches). each byte stored in the computer has it's own memory location which has an address. (which is the only way the computer could ever find it again.)

Nibble=half a byte, or the sub-group of four

Bit=a single switch.

So a computer that has 48K (K=approximately 1000) has a total memory capacity of approximately 48,000 bytes. (You can figure that out in nibbles and bits.) For the precise: K actually equals 1024.

The computer uses one byte for each number or character it stores. As you remember, there is a standard system for storing letters and special characters (which is fortunate, otherwise every microcomputer maker would have developed a different coding system.) This code is called the American Standard Code for Information (or old fried ASCII). For example

| you type | ASCII Decimal | ASCII hexidecimal | binary |
|---|---|---|---|
| J | 74 | 4A | 01001010 |
| I | 73 | 49 | 01001001 |
| M | 77 | 4D | 01001101 |
| | | | |
| 2 | 50 | 32 | 00110010 |
| + | 43 | 2B | 00101011 |
| 3 | 51 | 33 | 00110011 |
| = | 61 | 3D | 00110101 |
| 5 | 53 | 35 | 00110101 |

(Numeric variables, numbers, are stored differently. Each number takes 5 bytes)
In the early days of computers, programmers had to program the machine directly in binary. "Higher level languages" were developed so that we can use shorthand (commands, numbers, characters) which the languages then translate for us. Bless the language developers.

C. Flow Charting

In earlier lessons, you have been asked to make charts to follow loops in your programs. There is another type of charting which is used to

help you follow the logic of your programming. In the days when flow charting used to be required of every programming student, some students used the chart when they were developing the program, some wrote it after the fact just to hand in (rather similar to requiring outlines for writing assignments.)

Flow charting is an interesting technique and can help you:

1. try out the logic of a program you developing
2. keep track of a program you are writing
3. compare alternative approaches to a problem with others.
4. make you look like you know what you are doing.

There are a few standard symbols:

start or stop (very useful)

operation (something is being done)

input (data from user,program or file)
or output (to monitor or printer)

decision (a really nice visual symbol for choice)

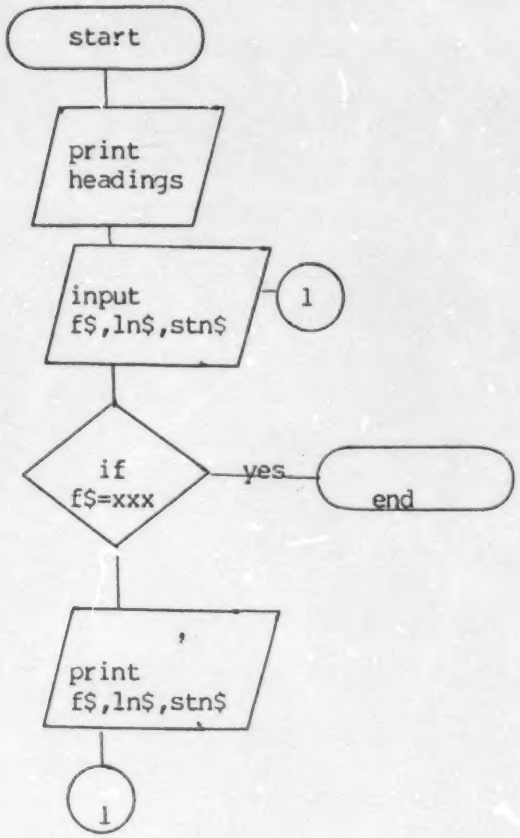
choices are written on lines,
no is traditionally at bottom

connector, reduces number of lines

go to, connection between other symbols

Charts traditionally begin at the top of the page and flow down. Decision lines can go to both sides, but most work is shown on the right. (For those of you who use PERT Charts for program management, you will now understand that PERT was developed by computer people. And those of you familiar with electrical engineering, will realize that it all came from there.)

An example of a flowchart for a program we have done (Lesson 3) would be:




As you can see, it is relatively easy from this chart to understand the logic of the program. However, you would need the program itself if wanted to copy or make changes in the program.

Flow charting is also an opportunity for people who don't normally work with visual representations of decision-making systems (social studies and philosophy books are full of them) to use such a system in situations that have personal meaning. There are also modified flow-charting forms, one very nice one is called pseudo-code and is very well described in Richard Haskell's APPLE BASIC, Prentice Hall, 1982.

REVIEW

TERMS

Sequential Access Files

Files which are read by the computer from the beginning to the end. Very economical of storage space, very difficult to change.

Random Access Files

Files which set aside a predetermined number of bytes per record. Wasteful of storage space, very easy to change.

BINARY

Base 2 number system (0,1) which is used to describe the configuration of switches in the computer. Switches are either off, low current, (0) or on, high current, (1).

HEXIDECIMAL

Base 16 number system (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) which is used to describe the combination of on/off positions of 4 switches (a nibble). The advantage of hexidecimal is that only one space is used (the symbol after 9 is A, not 10) to represent the 16 possible switch combinations of the four switches. To show you are a computernik, always call it hex, never hexidecimal.

BIT

One on/off switch

NIBBLE

Four on/off switches

BYTE

Eight on/off switches (two nibbles). This is the basic character translation unit of the computer. Each ASCII code character takes one byte.

K

1024 bytes. Most frequently used to describe the memory capacity (usually the Random Access Memory) of the computer.

FLOW CHART

Visual representation of any decision-making process. Excellent way to work on the logic of a computer program. There are a few standard symbols which are used to help you take advantage of the power of this

system.

COMMANDS

DOS=CHR$(4)

Sets the value of DOS$ to ctrl/D. The ctrl/D combination must be used at the beginning of any command to the files.

PRINT DOS$;"OPEN STU1,L200

The first two words tell the computer this is a command to the files; open says to locate and get ready to do something with (or start) a file name STU1 in which each record has 200 bytes.

PRINT DOS$;"WRITE STU1,R";17

Also a command to the files. The computer is told to enter the information (probably in the next PRINT statement) into the random access file named STU1 into record #17.

PRINT DOS$;"READ STU1,R";21

Another command to the files. Asks the computer to get data (probably in the next INPUT statement) from record number 21 in the random access file entitled STU1.

PRINT DOS$;"CLOSE STU1"

Saves the file STU1. Without this command, the file would be destroyed when the computer was turned off or the program rerun.

PRINT DOS$

If the computer has not been working with files, tells it to begin; if it has been working with files, tells it to stop. it WILL NOT save the file for future use.

QUIZ on LESSON 8

random access files

sequential access files

binary

hex

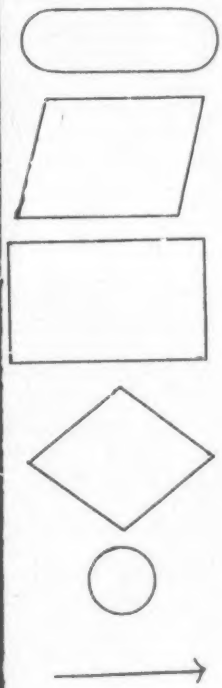
byte

bit

nibble

K

```
D$=CHR$(4)
PRINT D$;"OPEN ART,L20"
PRINT D$;"READ ART,R";75
PRINT D$;"WRITE ART,R";23
PRINT D$;"CLOSE ART"
PRINT D$
```



HAMPSHIRE EDUCATIONAL COLLABORATIVE: TRAINING MANUALS

INSERVICE SERIES

| | |
|---|---|
| Helping Teachers Become Inservice Facilitators | 12.00 |
| Nutrition and You in a Month or Two | 12.00 |
| The N.E.W. Blue Book of Nutrition Education | 10.00 |
| Developing an Energy Education Curriculum | 5.00 |
| Producing a Small Nonprofit Newsletter | 5.00 |

COMPUTER EDUCATION INSERVICE SERIES

| | |
|---|---|
| Introduction to Microcomputers | 5.00 |
| Developing a Computer Curriculum | 5.00 |
| Helping Teachers Become Computer Literate: A BASIC Approach | 6.00 |
| Introducing Logo | 5.00 |
| Word Processing for the Author | 3.00 |
| An Intensive Look at Word Processing: PIE Writer Workbook | 3.00 |
| Using Visicalc and Visiplot | 3.00 |
| Databases: Computerized Resource Retrieval Systems | 3.00 |
| Using Your Epson Printer | 3.00 |

Training manuals for specific word processing, spreadsheet and database programs (including WordStar, Bank Street Writer, Magic Window, Supercalc, dBASE, Notebook and pfs Write and Report) are also available for $1.00 each.

COMPUTER EDUCATION STUDENT SERIES

| | |
|---|---|
| Elementary BASIC Programming Activities | 6.00 |
| BASIC Programming Activities for Secondary Students | 7.00 |
| Project Logic Handbook: Computer Literacy Through BASIC (contains elementary and secondary manuals plus curriculum guides in loose-leaf binder) | 18.00 |

Order from: The Hampshire Educational Collaborative
The Center School, South Hadley, MA 01075



BEST COPY AVAILABLE